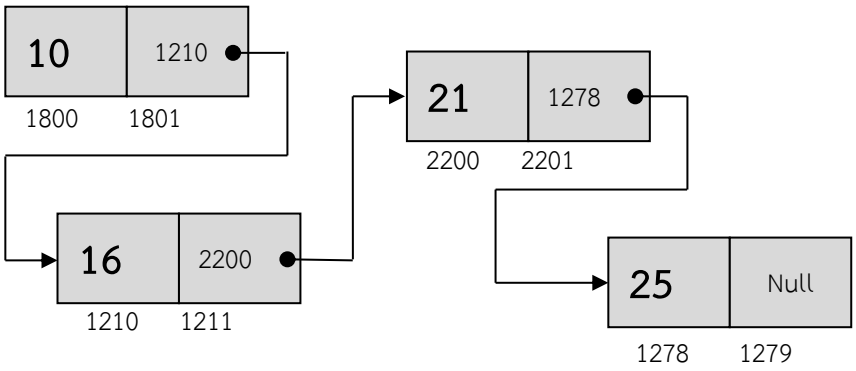
1. ลิงค์ลิสต์ (Linked List)

ลิงค์ลิสต์เป็นโครงสร้างข้อมูลที่จัดเก็บข้อมูลที่มีความสัมพันธ์กัน โดยข้อมูลแต่ละตัวเรียกว่า “โหนด (node)” ซึ่งถูกจัดเก็บในพื้นที่หน่วยความจำที่เรียงต่อเนื่องกันหรือไม่เรียงต่อเนื่องกันก็ได้

โหนดของลิงค์ลิสต์สามารถจัดเก็บด้วยข้อมูลแบบ `struct` ที่ประกอบด้วยค่าข้อมูล และตัวชี้ (pointer) เพื่อบอกตำแหน่งจัดเก็บของโหนดถัดไป ทำให้สามารถเก็บข้อมูลในหน่วยความจำที่ไม่เรียงต่อเนื่องกันได้ มีตัวอย่างดังรูปที่ 1.1 และตัวอย่างของโหนดเมื่อประกาศข้อมูลเป็น `struct` เป็นดังนี้

```
struct nodes
{
   int data;
   struct nodes *next;
};
```



รูปที่ 1.1 โครงสร้างข้อมูลแบบลิงค์ลิสต์ ที่จัดเก็บในหน่วยความจำที่ไม่เรียงต่อเนื่องกัน

การทำงานกับลิงค์ลิสต์ ประกอบด้วย

1. **การค้นหาข้อมูลโหนด (searching)** สามารถทำได้โดยการค้นหาแบบเรียงลำดับ (sequential search) คือ มีการพิจารณาจากโหนดแรกจนถึงกระทั่งเจอโหนดที่ต้องการหรือโหนดสุดท้ายของลิงค์ลิสต์

2. **การเพิ่มโหนด (insertion)** ในการเพิ่มโหนดในลิงค์ลิสต์ ที่มีพอยน์เตอร์ head ชี้ที่โหนดแรก นอกจากนี้จะต้องมีการระบุว่าต้องการเพิ่มที่ตำแหน่งใด มีขั้นตอนการทำงานดังนี้

กรณีเพิ่มหลังโหนด A

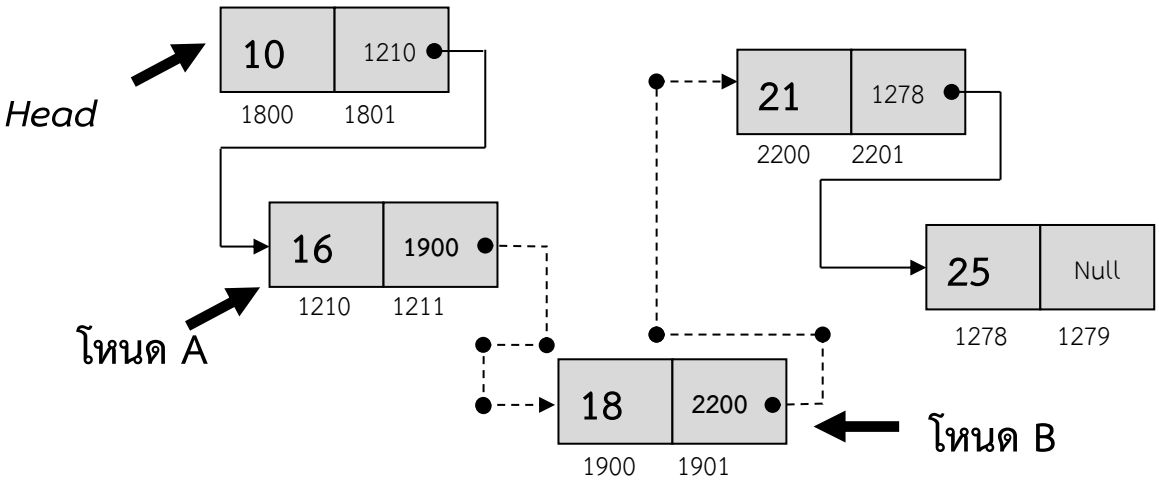
a. จองเนื้อที่ในหน่วยความจำพร้อมทั้งกำหนดค่าเริ่มต้นสำหรับโหนดใหม่ (โหนด B)

b. ค้นหาตำแหน่งที่ต้องการแทรกโหนดใหม่ โดยเริ่มค้นจากโหนดแรกที่ถูกชี้ด้วยพอยน์เตอร์ Head ในที่นี้ให้แทรกโหนดใหม่หลังโหนด A

c. เก็บตำแหน่งที่พอยน์เตอร์ชี้โหนดถัดไปของโหนด A ไว้

d. ให้พอยน์เตอร์ที่ชี้โหนดถัดไปของโหนด A ชี้ไปยังโหนดใหม่ (โหนด B)

e. ให้พอยน์เตอร์ที่ชี้โหนดถัดไปของโหนด B ชี้ไปตำแหน่งที่เก็บไว้ในข้อ c



รูปที่ 1.2 โครงสร้างข้อมูลแบบลิงค์ลิสต์ หลังเพิ่มโหนดใหม่ในลิงค์ลิสต์ของรูปที่ 1.1

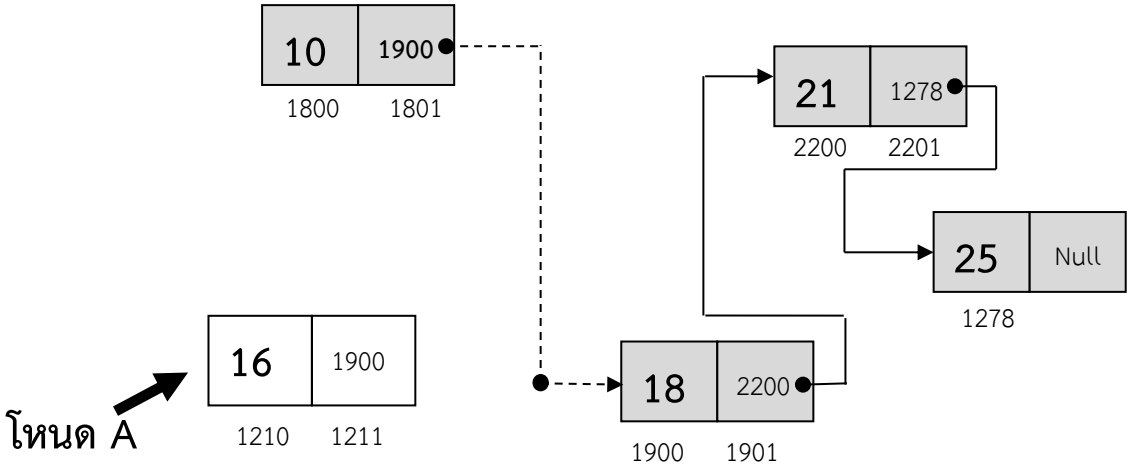
3. **การลบโหนด (Deletion)** เป็นการเปลี่ยนพอนย์เตอร์ที่ชี้โหนดที่ต้องการลบไปชี้โหนดที่โหนดที่ต้องการลบชี้

กรณีลบโหนด A

a. ค้นหาโหนดที่ต้องการลบ โดยเริ่มค้นจากโหนดแรกที่ถูกชี้ด้วยพอยน์เตอร์ *head* (ในที่นี้ต้องการลบโหนด A)

b. เปลี่ยนพอนย์เตอร์ที่ชี้โหนด A ไปชี้ยังโหนดที่โหนด A ชี้แทน

c. คืนหน่วยความจำของโหนด A



รูปที่ 1.3 โครงสร้างข้อมูลแบบลิงค์ลิสต์ของรูปที่ 1.2 หลังลบโหนด A

Standard Template Library (STL)

STL คือ library มาตรฐานของภาษา C++ ที่สร้างขึ้นมาเพื่อทำงานหรือจัดการกับข้อมูลประเภทโครงสร้างข้อมูล เช่น list, queue, stack, vector เป็นต้น

การประกาศการใช้ list และฟังก์ชันที่ใช้

Operation	Description
`list<type> c`	Creates an empty list without any elements
`c.size()`	Returns the actual number of elements
`c.front()`	Returns the first element (no check whether a first element exists)
`c.back()`	Returns the last element (no check whether a last element exists)
`c.insert(pos, elem)`	Inserts at iterator position `pos` a copy of `elem` and returns the position of the new element
`c.push_back(elem)`	Appends a copy of `elem` at the end
`c.push_front(elem)`	Inserts a copy of `elem` at the beginning
`c.pop_back()`	Removes the last element (does not return it)
`c.pop_front()`	Removes the first element (does not return it)
`c.erase(pos)`	Removes the element at iterator position `pos` and returns the position of the next element
`c.sort()`	Sorts all elements with operator <
`c.reverse()`	Reverses the order of all elements
`c.begin()`	Return iterator to beginning
`c.end()`	Return iterator to end

ตัวอย่างที่ 1.1 การสร้าง list และแสดงค่าใน list (ต้องกำหนดการคอมไพล์ด้วย c++11)

```
1.   #include <iostream>
2.
3.   #include <list>
4.   using namespace std;
5.   void printLists1 (list <int> ls);
6.   void printLists2 (list <int> ls);
7.
8.   int main() {
9.      int i;
10.     list<int> list1, list2;
11.     for (i=0; i<6; ++i)
12.          list1.push_back(i);
13.     printLists1(list1);
14.     printLists2(list1);
15.     return 0;
16.  }
17.
18.  void printLists1 (list <int> ls) {
19.     list<int>::iterator it;
20.     for (it=ls.begin();it!=ls.end();it++)
21.          cout<< *it << " ";
22.     cout<<endl;
23.  }
24.  void printLists2 (list <int> ls) {
25.     for (auto it : ls)
26.        cout<< it << " ";
27.     cout<<endl;
28.  }
```

ผลลัพธ์

```
0 1 2 3 4 5
0 1 2 3 4 5
```

ตัวอย่างที่ 1.2 การใช้ push_front(), insert() และ sort()

```
1.   #include <iostream>
2.   #include <list>
3.   using namespace std;
4.   void printLists (list <int> ls);
5.
6.   int main() {
7.      int i;
8.      list<int> list1;
9.      list<int>::iterator it;
10.
11.     for (i=0; i<6; i++)
12.          list1.push_front(i);
13.     printLists(list1);
14.
15.     it = list1.begin();
16.     advance(it, 2);
17.     list1.insert(it,99);
18.     printLists(list1);
19.
20.     list1.sort();
21.     printLists(list1);
22.     return 0;
23.  }
24.
25.  void printLists (list <int> ls) {
26.      for (auto it : ls)
27.        cout<< it << " ";
28.     cout<<endl;
28.
30.  }
```

ผลลัพธ์

```
5 4 3 2 1 0
5 4 99 3 2 1 0
0 1 2 3 4 5 99
```

ตัวอย่างที่ 1.3 การเปลี่ยนแปลงค่าใน list

```
1.   #include <iostream>
2.   #include <list>
3.   using namespace std;
4.   void printLists (list <int> ls);
5.
6.   int main() {
7.      int i;
8.      list<int> ls;
9.      for (i=0; i<6; ++i)
10.       ls.push_back(i);
11.     printLists(ls);
12.     list<int>::iterator it;
13.     for (it=ls.begin();it!=ls.end();it++)
14.      *it = *it+5;
15.     cout << "***Lists after changing***\n";
16.     printLists(ls);
17.     return 0;
18.  }
19.
20.  void printLists (list <int> ls) {
21.     list<int>::iterator it;
22.     for (it=ls.begin();it!=ls.end();it++)
23.       cout<< *it << " ";
24.     cout<<endl;
25.  }
```

ผลลัพธ์

```
0 1 2 3 4 5
***Lists after changing***
5 6 7 8 9 10
```

ตัวอย่างที่ 1.4 การใช้ list กับ struct ใน C++ (ให้ลองปรับโปรแกรมโดยทำให้สามารถเรียงได้ตาม amount)

```
1.  #include <iostream>
2.  #include <list>
3.  using namespace std;
4.  struct Node {
5.      char name[30];
6.      int amount;
7.  };
8.  bool compareByName(const Node left, const Node right){
9.     if(left.name <= right.name)
10.        return true;
11.    else
12.        return false;
13. }
14. bool compareByAmount(const Node left, const Node right){
15. // ปรับโปรแกรมลงในส่วนนี้
16.
17.
18.
19. }
20. void printLists (list <Node> ls) {
21.     for (auto it : ls)
22.       cout<< it.name << " " << it.amount << endl;
23.    cout<<endl;
24. }
25. int main() {
26.    Node data;
27.    list <Node> ls;
28.    data = Node{"melon", 5};
29.    ls.push_back(data);
30.    data = Node{"cherry", 15};
31.    ls.push_back(data);
32.    data = Node{"apple", 10};
33.    ls.push_back(data);
34.    printLists(ls);
35.    //---- Order by name ------------
36.    ls.sort(compareByName);
37.    printf("\n***Order by name***\n");
38.    printLists(ls);
39.    //---- Order by amount ------------
40.    // ปรับโปรแกรมลงในส่วนนี้ ให้เรียงตาม amount
41.
42.
43.
44.
46.
47.    return 0;
48. }
```

ผลลัพธ์

```
melon 5
cherry 15
apple 10

***Order by name***
apple 10
cherry 15
melon 5
```

ตัวอย่างที่ 1.5 การเปลี่ยนแปลงค่าใน list กับ struct

```
1.  #include <iostream>
2.  #include <list>
3.  using namespace std;
4.
5.  struct Node {
6.      string name;
7.      float amount;
8.  };
9.
10. bool compareByName(const Node left, const Node right){
11.    if(left.name <= right.name)
12.        return true;
13.    else
14.        return false;
15. }
16.
17. void printLists (list <Node> ls) {
18.     for (auto it : ls)
19.       cout<< it.name << " " << it.amount << endl;
20.    cout<<endl;
21. }
22.
23. int main() {
24.    Node data;
25.    list <Node> ls;
26.
27.    data = Node{"melon", 5};
28.    ls.push_back(data);
29.
30.    data = Node{"cherry", 15};
31.    ls.push_back(data);
32.
33.    data = Node{"apple", 10};
34.    ls.push_back(data);
35.
```

```
36.    printLists(ls);
37.    list<Node>::iterator it;
38.    for (it=ls.begin();it!=ls.end();it++)  {
39.      if (it->amount > 5)  {
40.        it->amount = it->amount * 0.1;
41.        it->name = "sale";
42.      }
43.    }
44.    cout << "***Lists after changing***\n";
45.    printLists(ls);
46.    return 0;
47.  }
```

ผลลัพธ์

```
melon 5
cherry 15
apple 10

***Lists after changing***
melon 5
sale 1.5
sale 1
```